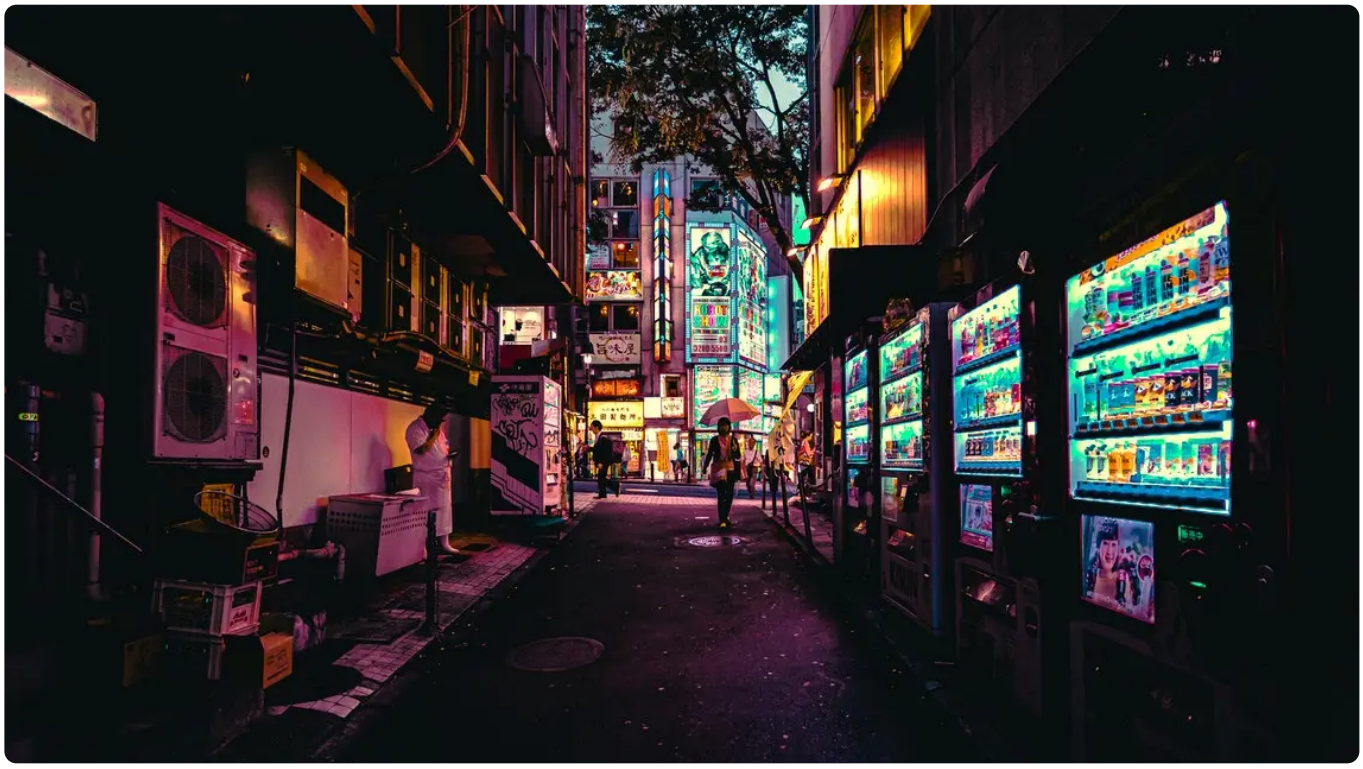
下载APP

10｜变量声明：静态语言有别于动态语言的重要特征

2021-11-03 Tony Bai

《Tony Bai · Go语言第一课》 课程介绍 >



讲述：Tony Bai
时长 17:01 大小 15.60M

你好，我是 Tony Bai。

今天我们将深入 Go 语法细节，学习静态语言有别于动态语言的一个重要特征：**变量声明**。那么变量声明究竟解决的是什么问题呢？我们先从变量这个概念说起。

在编程语言中，为了方便操作内存特定位置的数据，我们用一个特定的名字与位于特定位置的内存块绑定在一起，这个名字被称为**变量**。

但这并不代表我们可以通过变量随意引用或修改内存，**变量所绑定的内存区域是要有明确的边界的**。也就是说，通过这样一个变量，我们究竟可以操作 4 个字节内存还是 8 个字节内存，又或是 256 个字节内存，编程语言的编译器或解释器需要明确地知道。

领资料

那么，编程语言的编译器或解释器是如何知道一个变量所能引用的内存区域边界呢？

其实，动态语言和静态语言有不同的处理方式。动态语言（比如 Python、Ruby 等）的解释器可以在运行时通过对变量赋值的分析，自动确定变量的边界。并且在动态语言中，一个变量可以在运行时被赋予大小不同的边界。

而静态编程语言在这方面的“体验略差”。静态类型语言编译器必须明确知道一个变量的边界才允许使用这个变量，但静态语言编译器又没能力自动提供这个信息，这个边界信息必须由这门语言的使用者提供，于是就有了“**变量声明**”。通过变量声明，语言使用者可以显式告知编译器一个变量的边界信息。在具体实现层面呢，这个边界信息由变量的类型属性赋予。

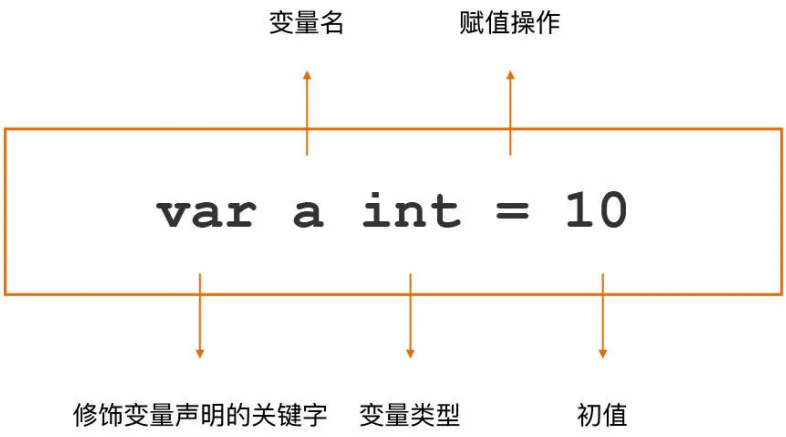
作为身处静态编程语言阵营的 Go 语言，它沿袭了静态语言的这一要求：**使用变量之前需要先进行变量声明**。

首先，让我们先来看看 Go 语言的变量声明方法。

Go 语言的变量声明方法

我们前面说过，Go 是静态语言，所有变量在使用前必须先进行声明。声明的意义在于告诉编译器该变量可以操作的内存的边界信息，而这种边界通常又是由变量的类型信息提供的。

在 Go 语言中，有一个通用的变量声明方法是这样的：



这个变量声明分为四个部分：

var 是修饰变量声明的关键字；

a 为变量名；

int 为该变量的类型；

10 是变量的初值。

你看啊，其实 Go 语言的变量声明形式与其他主流静态语言有一个显著的差异，那就是**它将变量名放在了类型的前面**。这样做有什么好处呢？我先不说，我想请你思考一下。这个类型为变量提供了边界信息，在 Go 语言中，无论什么类型的变量，都可以使用这种形式进行变量声明。

但是，如果你没有显式为变量赋予初值，Go 编译器会为变量赋予这个类型的零值：

复制代码

```
1 var a int // a的初值为int类型的零值：0
```

什么是类型的零值呢？Go 语言的每种原生类型都有它的默认值（这些原生类型我们后面再讲），这个默认值就是这个类型的零值。这里我给你写了 Go 规范定义的内置原生类型的默认值（即零值）：

内置原生类型	默认值（零值）
所有整型类型	0
浮点类型	0.0
布尔类型	FALSE
字符串类型	""
指针、接口、切片、channel、map和函数类型	nil

极客时间

另外，像数组、结构体这样复合类型变量的零值就是它们组成元素都为零值时的结果。

除了单独声明每个变量外，Go 语言还提供了变量声明块（block）的语法形式，可以用一个 var 关键字将多个变量声明放在一起，像下面代码这样：

复制代码

```
var (
    a int = 128
    b int8 = 6
    s string = "hello"
    c rune = 'A'
    t bool = true
)
```

你看在这个变量声明块中，我们通过一个 var 关键字声明了 5 个不同类型的变量。而且，Go 语言还支持在一行变量声明中同时声明多个变量：

复制代码

```
var a, b, c int = 5, 6, 7
```

这样的多变量声明同样也可以用在变量声明块中，像下面这样：

复制代码

```
var (
    a, b, c int = 5, 6, 7
    c, d, e rune = 'C', 'D', 'E'
)
```

当然了，虽然我们现在写的多变量声明都是在声明同一类型的变量，但是它也适用于声明不同类型的变量，这个我们等会儿会详细讲讲。

除了上面这种通用的变量声明形式，为了给开发者带来更好的使用体验，Go 语言还提供了两种变量声明的“语法糖”，下面我们逐一来学习一下。

1. 省略类型信息的声明：

在通用的变量声明的基础上，Go 编译器允许我们省略变量声明中的类型信息，它的标准范式是“**var varName = initExpression**”，比如下面就是一个省略了类型信息的变量声明：

```
1 var b = 13
```

复制代码

那么 Go 编译器在遇到这样的变量声明后是如何确定变量的类型信息呢？

其实很简单，Go 编译器会根据右侧变量初值自动推导出变量的类型，并给这个变量赋予初值所对应的默认类型。比如，整型值的默认类型 int，浮点值的默认类型为 float64，复数值的默认类型为 complex128。其他类型值的默认类型就更好分辨了，在 Go 语言中仅有唯一与之对应的类型，比如布尔值的默认类型只能是 bool，字符值默认类型只能是 rune，字符串值的默认类型只能是 string 等。

如果我们不接受默认类型，而是要显式地为变量指定类型，除了通用的声明形式，我们还可以通过显式类型转型达到我们的目的：

```
1 var b = int32(13)
```

复制代码

显然这种省略类型信息声明的“语法糖”仅**适用于在变量声明的同时显式赋予变量初值的情况**，下面这种没有初值的声明形式是不被允许的：

```
1 var b
```

复制代码

结合多变量声明，我们可以使用这种变量声明“语法糖”声明多个不同类型的变量：

```
1 var a, b, c = 12, 'A', "hello"
```

复制代码

在这个变量声明中，我们声明了三个变量 a、b 和 c，但它们分别具有不同的类型，分别为 int、rune 和 string。

在这种变量声明语法糖中，我们省去了变量类型信息，但 Go 编译器会为我们自动推导出类型信息。那是否还有更简化的变量声明形式呢？答案是有的。下面我们就来看看短变量声明。

2. 短变量声明：

其实，Go 语言还为我们提供了最简化的变量声明形式：**短变量声明**。使用短变量声明时，我们甚至可以省去 var 关键字以及类型信息，它的标准范式是“varName := initExpression”。我这里也举了几个例子：

```
1 a := 12
2 b := 'A'
3 c := "hello"
```

复制代码

这里我们看到，短变量声明将通用变量声明中的四个部分省去了两个，但它并没有使用赋值操作符“=”，而是使用了短变量声明专用的“:=”。这个原理和上一种省略类型信息的声明语法糖一样，短变量声明中的变量类型也是由 Go 编译器自动推导出来的。

而且，短变量声明也支持一次声明多个变量，而且形式更为简洁，是这个样子的：

```
1 a, b, c := 12, 'A', "hello"
```

复制代码

不过呢，短变量声明的使用也是有约束的，并不是所有变量都能用短变量声明来声明的，这个你会在下面的讲解中了解到。

好了，现在我们已经学习了至少三种变量声明形式了。这时候你可能有些犯迷糊了：这些变量声明形式是否适合所有变量呢？我到底该使用哪一种呢？别急，在揭晓答案之前，我们需要学习点预备知识：**Go 语言的两类变量**。

通常来说，Go 语言的变量可以分为两类：一类称为**包级变量 (package varible)**，也就是在包级别可见的变量。如果是导出变量（大写字母开头），那么这个包级变量也可以被视为全局变量；另一类则是**局部变量 (local varible)**，也就是 Go 函数或方法体内声明的变量，仅在函数或方法体内可见。而我们声明的所有变量都逃不开这两种。

有了这个预备知识，接下来我们就来分别说明一下这两类变量在声明形式选择上的方法，以及一些最佳实践。

包级变量的声明形式

首先，我先下个结论：**包级变量只能使用带有 var 关键字的变量声明形式，不能使用短变量声明形式，但在形式细节上可以有一定灵活度。**具体这个灵活度怎么去考虑呢？我们可以从“变量声明时是否延迟初始化”这个角度，对包级变量的声明形式进行一次分类。

第一类：声明并同时显式初始化。

你先看看这个代码：

复制代码

```
// $GOROOT/src/io/io.go
var ErrShortWrite = errors.New("short write")
var ErrShortBuffer = errors.New("short buffer")
var EOF = errors.New("EOF")
```

我们可以看到，这个代码块里声明的变量都是 io 包的包级变量。在 Go 标准库中，对于变量声明的同时进行显式初始化的这类包级变量，实践中多使用这种省略类型信息的“语法糖”格式：

```
var varName = initExpression
```

复制代码

就像我们前面说过的那样，Go 编译器会自动根据等号右侧 InitExpression 结果值的类型，来确定左侧声明的变量的类型，这个类型会是结果值对应类型的默认类型。

当然，如果我们不接受默认类型，而是**要显式地为包级变量指定类型**，那么我们有两种方式，我这里给出了两种包级变量的声明形式的对比示例。

复制代码

```
//第一种：
plain
var a = 13 // 使用默认类型
var b int32 = 17  // 显式指定类型
var f float32 = 3.14 // 显式指定类型

//第二种：
var a = 13 // 使用默认类型
var b = int32(17) // 显式指定类型
var f = float32(3.14) // 显式指定类型
```

虽然这两种方式都是可以使用的，但从声明一致性的角度出发，Go 更推荐我们使用后者，这样能统一接受默认类型和显式指定类型这两种声明形式，尤其是在将这些变量放在一个 var 块中声明时，你会更明显地看到这一点。

所以我们更青睐下面这样的形式：

复制代码

```
var (
  a = 13
  b = int32(17)
  f = float32(3.14)
)
```

而不是下面这种看起来不一致的声明形式：

复制代码

```
var (
```

```
    a = 13
    b int32 = 17
    f float32 = 3.14
)
```

第二类：声明但延迟初始化。

对于声明时并不立即显式初始化的包级变量，我们可以使用下面这种通用变量声明形式：

复制代码

```
var a int32
var f float64
```

我们知道，虽然没有显式初始化，Go 语言也会让这些变量拥有初始的“零值”。如果是自定义的类型，我也建议你尽量保证它的零值是可用的。

这里还有一个注意事项，就是声明聚类与就近原则。

正好，Go 语言提供了**变量声明块**用来把多个的变量声明放在一起，并且在语法上也不会限制放置在 var 块中的声明类型，那我们就应该学会充分利用 var 变量声明块，让我们变量声明更规整，更具可读性，现在我们就来试试看。

通常，我们会将同一类的变量声明放在一个 var 变量声明块中，不同类的声明放在不同的 var 声明块中，比如下面就是我从标准库 net 包中摘取的两段变量声明代码：

复制代码

```
// $GOROOT/src/net/net.go

var (
    netGo  bool
    netCgo bool
)

var (
    aLongTimeAgo = time.Unix(1, 0)
    noDeadline = time.Time{}
    noCancel   = (chan struct{})(nil)
)
```

我们可以看到，上面这两个 var 声明块各自声明了一类特定用途的包级变量。那我就要问了，你还能从中看出什么包级变量声明的原则吗？

其实，我们可以将延迟初始化的变量声明放在一个 var 声明块 (比如上面的第一个 var 声明块)，然后将声明且显式初始化的变量放在另一个 var 块中（比如上面的第二个 var 声明块），这里我称这种方式为**“声明聚类”**，声明聚类可以提升代码可读性。

到这里，你可能还会有一个问题：我们是否应该将包级变量的声明全部集中放在源文件头部呢？答案不能一概而论。

使用静态编程语言的开发人员都知道，变量声明最佳实践中还有一条：**就近原则**。也就是说我们尽可能在靠近第一次使用变量的位置声明这个变量。就近原则实际上也是对变量的作用域最小化的一种实现手段。在 Go 标准库中我们也很容易找到符合就近原则的变量声明的例子，比如下面这段标准库 http 包中的代码就是这样：

复制代码

```
// $GOROOT/src/net/http/request.go

var ErrNoCookie = errors.New("http: named cookie not present")
func (r *Request) Cookie(name string) (*Cookie, error) {
    for _, c := range readCookies(r.Header, name) {
        return c, nil
    }
    return nil, ErrNoCookie
}
```

在这个代码块里，ErrNoCookie 这个变量在整个包中仅仅被用在了 Cookie 方法中，因此它被声明在紧邻 Cookie 方法定义的地方。当然了，如果一个包级变量在包内部被多处使用，那么这个变量还是放在源文件头部声明比较适合的。

接下来，我们再来看看另外一种变量：局部变量的声明形式。

局部变量的声明形式

有了包级变量做铺垫，我们再来讲解局部变量就容易很多了。和包级变量相比，局部变量又多了一种短变量声明形式，这是局部变量特有的一种变量声明形式，也是局部变量采用最多的一种声明形式。

这里我们也从“变量声明的时候是否延迟初始化”这个角度，对本地变量的声明形式进行分类说明。

第一类：对于延迟初始化的局部变量声明，我们采用通用的变量声明形式

其实，我们之前讲过的省略类型信息的声明和短变量声明这两种“语法糖”变量声明形式都不支持变量的延迟初始化，因此对于这类局部变量，和包级变量一样，我们只能采用通用的变量声明形式：

```
1 var err error
```

第二类：对于声明且显式初始化的局部变量，建议使用短变量声明形式

短变量声明形式是局部变量最常用的声明形式，它遍布在 Go 标准库代码中。对于接受默认类型的变量，我们使用下面这种形式：

```
1 a := 17
2 f := 3.14
3 s := "hello, gopher!"
```

对于不接受默认类型的变量，我们依然可以使用短变量声明形式，只是在":="右侧要做一个显式转型，以保持声明的一致性：

```
1 a := int32(17)
2 f := float32(3.14)
3 s := []byte("hello, gopher!")
```

这里我们还要注意：尽量在分支控制时使用短变量声明形式。

分支控制应该是 Go 中短变量声明形式应用得最广泛的场景了。在编写 Go 代码时，我们很少单独声明用于分支控制语句中的变量，而是将它与 if、for 等控制语句通过短变量声明

形式融合在一起，即在控制语句中直接声明用于控制语句代码块中的变量。

你看一下下面这个我摘自 Go 标准库中的代码，strings 包的 LastIndexAny 方法为我们很好地诠释了如何将短变量声明形式与分支控制语句融合在一起使用：

复制代码

```
// $GOROOT/src/strings/strings.go
func LastIndexAny(s, chars string) int {
    if chars == "" {
        // Avoid scanning all of s.
        return -1
    }
    if len(s) > 8 {
        // 作者注：在if条件控制语句中使用短变量声明形式声明了if代码块中要使用的变量as和isA
        if as, isASCII := makeASCIISet(chars); isASCII {
            for i := len(s) - 1; i >= 0; i-- {
                if as.contains(s[i]) {
                    return i
                }
            }
            return -1
        }
    }
    for i := len(s); i > 0; {
        // 作者注：在for循环控制语句中使用短变量声明形式声明了for代码块中要使用的变量c
        r, size := utf8.DecodeLastRuneInString(s[:i])
        i -= size
        for _, c := range chars {
            if r == c {
                return i
            }
        }
    }
    return -1
}
```

而且，短变量声明的这种融合的使用方式也体现出“就近”原则，让变量的作用域最小化。

另外，虽然良好的函数 / 方法设计都讲究“单一职责”，所以每个函数 / 方法规模都不大，很少需要应用 var 块来聚类声明局部变量，但是如果你在声明局部变量时遇到了适合聚类的应用场景，你也应该毫不犹豫地使用 var 声明块来声明多于一个的局部变量，具体写法你可以参考 Go 标准库 net 包中 resolveAddrList 方法：

复制代码

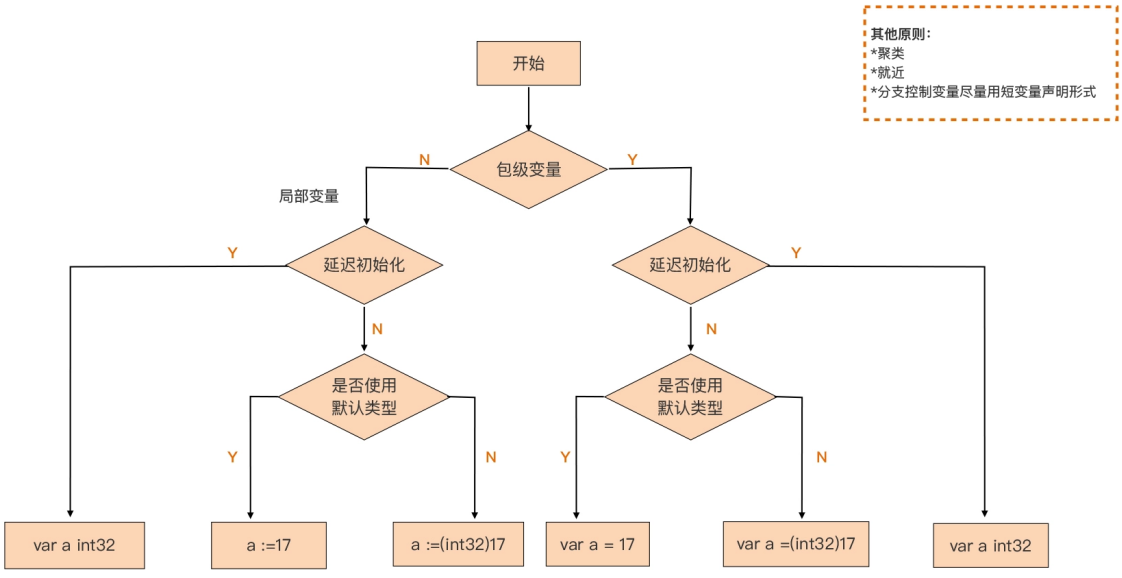
```
// $GOROOT/src/net/dial.go
func (r *Resolver) resolveAddrList(ctx context.Context, op, network,
                                   addr string, hint Addr) (addrList, error) {
    ... ...
    var (
        tcp      *TCPAddr
        udp      *UDPAddr
        ip       *IPAddr
        wildcard bool
    )
   ... ...
}
```

小结

好了，今天的课讲到这里就结束了，现在我们一起来回顾一下吧。

在这一讲中，我们学习了多种 Go 变量声明的方法，还学习了不同类型 Go 变量可以采用的变量声明形式和惯用法，以及一些变量声明的最佳实践原则。

具体来说，Go 语言提供了一种通用变量声明形式以及两种变量声明“语法糖”形式，而且 Go 包级变量和局部变量会根据具体情况选择不同的变量声明形式，这里我们用一幅图来做个形象化的小结：



你可以看到，良好的变量声明实践需要我们考虑多方面因素，包括明确要声明的变量是包级变量还是局部变量、是否要延迟初始化、是否接受默认类型、是否是分支控制变量并结合聚类和就近原则等。

说起来，Go 语言崇尚“做一件事只用一种方法”，但变量声明却似乎是一个例外。如果让 Go 语言的设计者重新来设计一次变量声明语法，我觉得他们很大可能不会给予开发们这么大的变量声明灵活性。作为开发者，我们要注意的是，在统一项目范围内，我们选择的变量声明的形式应该是一致的。

思考题

今天，我们的思考题是：与主流静态语言不同，在 Go 语言变量声明中，类型是放在变量名的后面的，你认为这样做有什么好处？欢迎在留言区给我留言。

感谢你和我一起学习，也欢迎你把这节课分享给更多对 Go 语言的类型声明感兴趣的朋友。我是 Tony Bai，我们下节课见。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 4 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 09 | 即学即练：构建一个Web服务就是这么简单

下一篇 11 | 代码块与作用域：如何保证变量不会被遮蔽？

精选留言 (13)

写留言

珅珅君
2021-11-03

https://blog.go-zh.org/gos-declaration-syntax 其实官方有给出解释，原因简单来说就是和C相比，在当参数是指针的复杂情况下，这种声明格式会相对好理解一点，链接里面有详细解释

作者回复: 正解。

1 3

i6u
2021-11-03

为了不一样而不一样？必要的写在最前面，不必要的可以省略的写在后面？

2

hhzzer
2021-11-03

想问一下老师，为啥go产生了这么多种变量声明方式？依照一个问题只用一种方法解决的原则，如果你重新设计，会选择保留哪一种申明方式呢？

展开

作者回复: 1. 保留这么多变量声明方式的原因，Go语言设计者们并未在公开场合说明过。这个我也无不能不负责任的揣测^_^。
2. rob pike只是说过，如果有机会从头设计go，他不会再提供这么多变量声明方式了。但没说要保留哪一种。如果是我设计，我可能会保留var a int以及短变量声明。

2

高雪斌
2021-11-03

就是既想要静态语言的严格，又想取动态语言的灵活的折中方案。

作者回复: 选Go就对了:)

1

jc9090kkk
2021-11-03

我认为类型放在变量名的后面是为了提高代码可读性的，我个人认为golang的设计者在设计go语言的时候大概率参考了c语言的语法，针对存在大量变量需要声明的场景下，go的变量类型放在变量名的后面可读性会高很多，不容易引起类型定义的混乱。

比如：...
展开

作者回复: ☑。答案非常细致。刚开始学go时，也不适应，后来适应后，就像你说的，在一些复杂函数/方法的声明中，类型在后面的方式更容易理解。

1

罗杰
2021-11-03

C 和 C++ 都是类型在前，变量名在后，记得非常清楚 CCMouse 老师在视频课上提到这才是正确的方法，但到现在其实我还没太明白原因。不过在 Rust 中采用了和 Go 相同的方式。所以希望老师解答一下原因
展开

作者回复: 其实官方早有正解，https://go.dev/blog/declaration-syntax 。我个人以前是C程序员，在读懂那些复杂函数的原型时，颇为费力。换做go之后，再未被这类问题困扰过:)

1

AlexWillBeGood

2021-11-03

老师好，请问“静态语言明确声明类型”会在编译阶段或运行阶段提高效率吗？

作者回复: 我觉得与效率无关。静态语言的类型声明是静态语言的机制所要求的。静态语言原本运行效率就是要比动态语言要高的，因为它已经编译为机器码了，无需开销较大的解释过程就可以运行了。

1

进化菌

2021-11-04

go的变量声明，方式还是挺灵活的。
把类型放在变量名后面，大概是更方便清晰的说明吧，毕竟前面已经有var关键字了~

展开

陈东

2021-11-03

从源头开始讲，这一讲很好，知所以然。

展开

aoe

2021-11-03

短短的几行代码没看懂……

展开

1

liaomars

2021-11-03

变量名放在前面应该是考虑代码的可读性才这样设计的吧

作者回复: 大约是如此。可以看看官方解释：https://go.dev/blog/declaration-syntax

大大~喜羊羊

2021-11-03

这么多声明语法确实有选择恐惧症了，就是不知道随心所欲地去声明变量后面会不会给自己挖坑

iLeGeND

2021-11-03

好处就是省略方便

展开